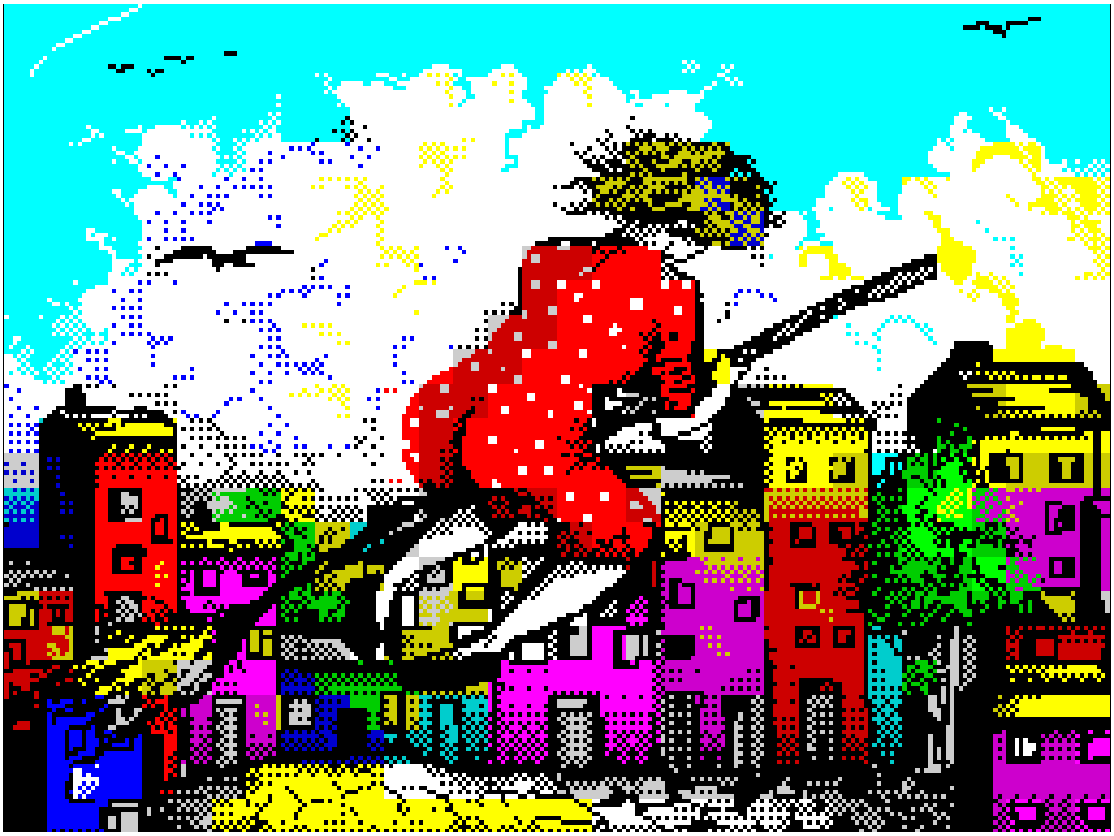
Für alle Spectrum- und SAM-Freunde

Spectrum & SAM Profi Club Köln



Dieses mal kein Flug zum Treffen in Wittenberg :(

Herausgeber und für den Inhalt verantwortlicher:
Leszek Chmielewski, Prager Straße 92/11/12, 1210 Wien, Österreich
@Mail: retrozx@gmail.com
Klubkonto (Inhaber: Bernhard Lutz):IBAN: DE59 5486 2500 0000 5461 43
SWIFT-Code: GENODE6K, BIC-Code: GENODE61SUW
KTO.: 546143, BLZ: 54862500 (VR Bank Südpfalz, Sitz: Landau)

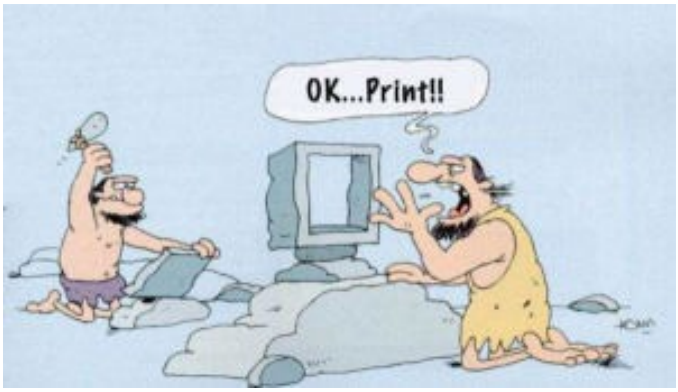
Ausgabe 234

3 Quartal 2013

Das Vorwort

http://www.womoteam.de/
http://spc.tlienhard.com/

Willkommen zu der Zeitschrift von Usern für User. Wir sind vor allem auf EURE Artikel angewiesen. Ich kann alleine keine (angepeilten) 24-32 Seiten füllen, so gerne ich es auch tun würde. Ehrenwort! Für eingeschickte Artikel gelten folgende Regeln:
Die Artikel müssen sich mit dem Spectrum, ZX81, SAM Coupé, Sprinter 2000 oder nahen Verwandten des Sinclair ZX Spectrum befassen, auch Artikel über passende Hardware und Software sind gerne gesehen.
MAC/PC Software: Nur wenn ausdrücklich direkt im Zusammenhang mit den eingangs erwähnten Computern. Sehr gerne: Crosscompiler, Emulatoren, Game Maker und dergleichen. Auf keinen Fall aber Remakes von Spielen alter Plattformen auf moderner Hardware.



Ausgabe 234 folgt schon ganz kurz nach Nr. 233. Tatsächlich habe ich an beiden Ausgaben parallel gearbeitet, denn anders wäre der Verzug nicht mehr einholbar gewesen. Es gab allerdings durch Verwechslungen ein paar „Unfälle“ wie z.B., dass die News gelöscht wurden. Ich sollte mich eben nicht so sehr stressen lassen, denn durch solche Probleme verzögert sich alles nur noch mehr und ich habe mehr Arbeit zu erledigen.

Ich hoffe, dass Euch die Artikel übers Programmieren etwas inspirieren um selbst neue Programme zu schreiben. Es erscheinen im Jahr mehr als 100 neue Spiele für den Spectrum. ca. 50% kommen aus Spanien, 30% aus England, 30% aus ehemaligen Ostblockländern und PI mal Daumen 2% aus dem deutschsprachigen Raum, und an diesen 2% bin ich stark beteiligt.
Falls das notwendige Crosscompiler-Werkzeug fehlt: Für Mac, Win und Linux gibt es z88dk ANSI-C) und ZXBC (erweiterter BASIC Compiler).
An einer ZXBC IDE arbeite ich und könnte sie in der Zukunft auch an z88dk anpassen.
Für SAM hingegen schaut es im Moment noch etwas trostlos aus: BASIC Compiler die sich sinnvoll benutzen lassen gibt es nicht, und z88dk hat keine vernünftigen Bibliotheken wie das SP1 für den Spectrum.

Da ich nun im Besitz eines Android Devices bin, kann ich in der Zukunft Programme (APPs=Android Platform Programs) mit Retro-Bezug dafür testen. Es gibt recht viele davon im Google Store. Interessante Programme sind u.a. Emulatoren (Marvin, Unreal Speccy) oder Hilfsprogramme (TapDancer, TeeZiX, Tapex). Wenn jemand noch andere hilfreiche Programme kennt, bitte um Informationen. Ich beiße nicht!

LCD-Leszek Chmielewski

Termine 2013+

04. -06.04.2014 : ZX-TEAM Treffen
D-36145, Hofbieber, (Mahlerts)
Die Hardwareprofis für den ZX81 und ZX-Spectrum mit originellen und innovativen Ideen!

23. -24.08.2014 : Spectrumania
Kreisvolkshochschule
Falkstraße 83
(Ecke Zimmermannstrasse)
06886 Lutherstadt Wittenberg

20. + 21.09.2014: Spectra-Joyce
Dorfgemeinschaftshaus
Bruchstrasse
D-34466 Nothfelden
(Wolfhagen)
Eine Initiative des Spectrum Profi Club und der Joyce User AG. Auf diesem Treffen stehen die Joyce-Rechner (CP/M) und die Sinclair Rechner im Mittelpunkt. Natürlich sind auch SAM Coupe und die Freunde weiterer 8Bit-Rechner willkommen!
Zusätzlich findet die Jahreshauptversammlung 2014 der Joyce-User-AG statt.

14-16.3.2014: Forever 2014 Olympic Edition.
Horna Suca 1059, Slovakia
48°57'55.908"N 17°59'14.460"E

Foreword-English

Welcome to the magazine by users for users. We are primarily dependent YOUR article. I alone can not fill (targeted) 24-32 pages, even I would like to do it. Word of honor! To be sent article, the following rules:

The articles have to deal with the Spectrum, ZX81, SAM Coupe, Sprinter 2000, or close relatives of the Sinclair ZX Spectrum, including articles on appropriate hardware and software are welcome.

Issue 234 follows quite soon after No. 233 In fact, I 've worked in parallel on two issues, because otherwise the delay would be too big to deal with it.
However, there was confusion by a few "accidents" such as that the news has been deleted. I was not meant to be so much stress, because by such problems, everything is delayed even more and I have to do more work.

I hope that you the articles about programming inspired someone to write new programs. Each year appear more than 100 new games for the Spectrum. about 50% come from Spain , 30% from England , 30% of the former Eastern Bloc countries and some 2% from the German-speaking countries, and on that 2% I am heavily involved.
If the necessary cross compiler tool is missing: For Mac, Win and Linux, there is z88dk ANSI -C) and ZXBC (Advanced BASIC compiler).

I'm working on ZXBC IDE and could adapt it also for z88dk in the future. For SAM, however, it looks at the moment kind of bleak: BASIC compiler which can be useful to use does not exist, and SAM has no reasonable z88dk libraries like the SP1 for the Spectrum.

Since I am in possession of an Android device now, I can test in the future programs (APPs = Android Platform Programs) with retro reference for it. There are quite a few of them in the Google Store. Interesting programs are, e.g., Emulators (Marvin, Unreal Speccy) or utilities (TapDancer, TeeZiX, Tapex). If anyone still knows other helpful programs , please send me some information. I will not bite!

LCD Leszek Chmielewski

Neuigkeiten für unseren „Alten“

I'm crawling in a dungeon

http://www.worldofspectrum.org/forums/showthread.php?t=44376

Kaum hat Felix „Spectral Dungeons“ fertiggestellt, schon hat er den Nachfolger fertiggestellt: „Escape from Cnossus“. Es spielt sich wesentlich besser als der Vorgänger.

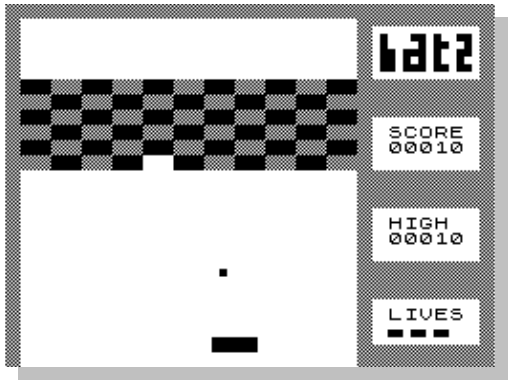
Der Fluch von Oddville

http://www.worldofspectrum.org/forums/showthread.php?t=44389

R-Tape hat seinen „Crap game Maker“ nun fertig und ein Spiel damit erstellt, das gar nicht mal so „Crap“ ist: „The Curse of Oddville“. Es ist ein Spiel der alten Machart (also 1984 oder so). Leider viel zu kurz...

Schlägertyp!

http://www.worldofspectrum.org/forums/showthread.php?t=44365



Batz ist ein Arkanoid-Klon von Peterz für ZX81 und 16K RAM, jetzt mit ZON-X Soundkarten-Unterstützung dank der Arbeit von Yerzmyey.

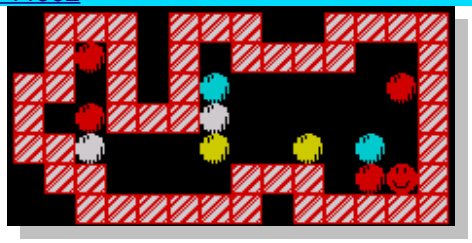
Doppelpack

http://www.worldofspectrum.org/forums/showthread.php?t=44390

Alessandro Grussu hat ein Pack namens „Al's Double Bill“ mit einer Neuauflage zwei seiner Spiele herausgebracht: „Lost in My Spectrum“ und „Apulija 13“. Beide wurden verbessert, z.B. durch mehr Levels, neue Sounds oder alternative Enden.

Morph!

http://www.worldofspectrum.org/forums/showthread.php?t=44332



TMS86 (Madsoft), ein C64

Programmierer, hat sich verbessert indem er Z80 Assembler lernte und eines seiner früheren Werke für den Spectrum umsetzte. So haben wir einen Puzzler mit zu Hires konvertierter Lowres Grafik, aber das stört nicht.
Jedenfalls hat ihm das Programmieren am Spectrum derart gefallen, dass er vorhat in Zukunft andere C64 Spiele von ihm auf den Spectrum umzusetzen.

Terry the Turtle Vector

http://www.worldofspectrum.org/infoseekid.cgi?id=0028158

Von „Terry the Turtle“ gibt es nun eine Sonderedition in Vektorgrafikdesign. Es bleibt aber immer noch ein Spiel in der Jet Set Willy Engine.

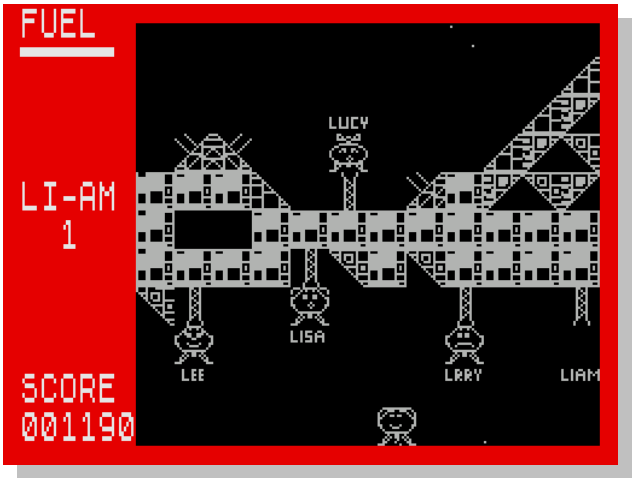
Skurff!

http://www.worldofspectrum.org/forums/showthread.php?t=44801

R-Tape wird nicht müde. Er hat mit seinem CGD das Spiel Skurff geschrieben. Wie bei Trap Door, muss man in dem Spiel die Wünsche seines Meisters erfüllen.

Larry the Lander II

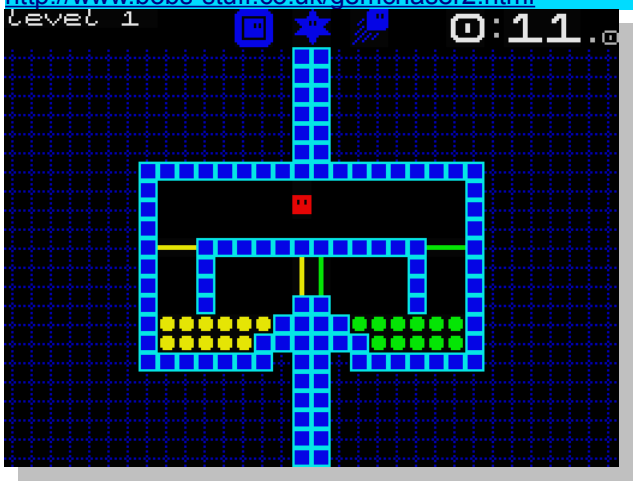
http://www.worldofspectrum.org/forums/showthread.php?t=44783



„Purple Unicorn“ (jammajup) hat den Nachfolger zu „Larry the Lander“ geschrieben: „Cosmic Carnage“. Jetzt ist die ganze Familie vertreten: Lisa Lander, Lucy Lander, Lee Lander,... Für jedes Familienmitglied gibt es ein eigenes Programm, und verschiedene Mitglieder haben unterschiedliche Eigenschaften die alle aus einer Vergleichsliste ersichtlich sind.

Jäger der Juwelen

http://www.bobs-stuff.co.uk/gemchaser2.html

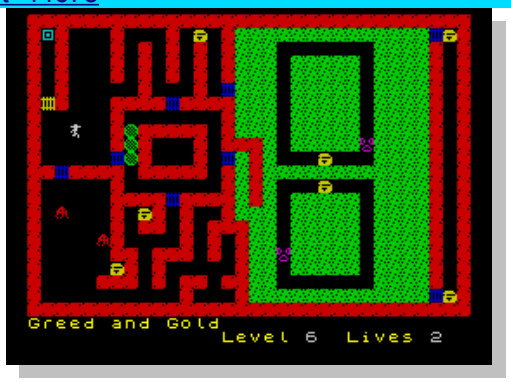


Bob hat sein Gem Chaser etwas

aufpoliert und als Gem Chaser II mit neuen Levels freigegeben.
70 Levels und eine faszinierende neue Spielidee. Wer kann da widerstehen? Ein Top-Titel!

Der verlorene Schatz

http://www.worldofspectrum.org/forums/showthread.php?t=44678



Rebelstar without a Cause hat „The Lost Treasure“ mit CGD geschrieben. Schön gemachtes Spiel für ein paar nette Stunden.

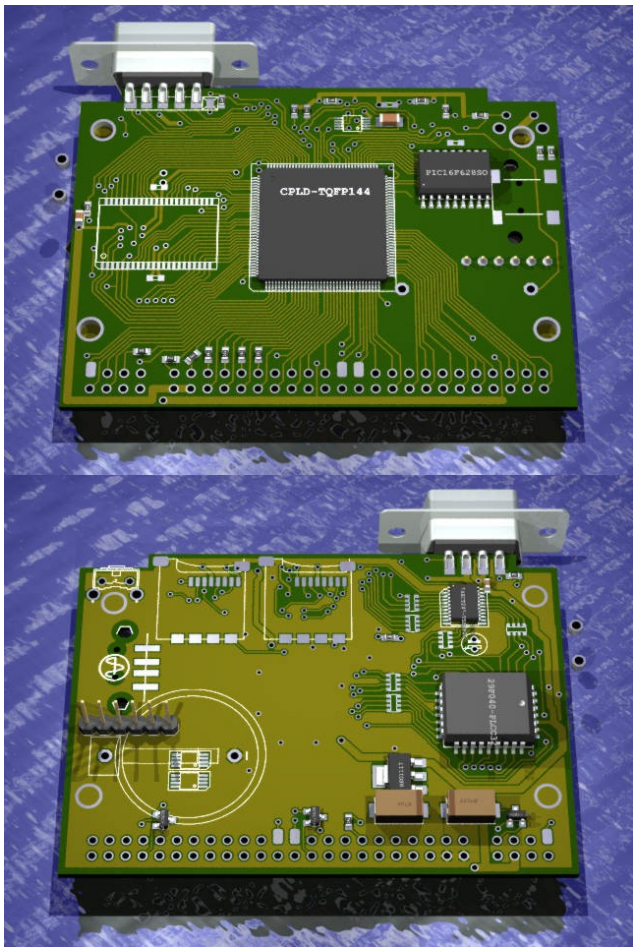
Umbenannt!

http://zxsandbox.blogspot.co.uk/

R-Tape (Dave Hughes) hat nun seinen C.G.D. fertiggestellt. Anfangs stand dies für „Crap Game Designer“, doch nach etlichen Verbesserungen wurde daraus der „Classic Game Designer“, der einem erlaubt Spiele zu entwerfen, die nach 1983 aussehen.
Es arbeitet mit Chars (7x7 Pixel Blöcken) die nicht pixelweise verschoben werden können, und erlaubt sehr schnelle Grafik.

Velesofts Universalinterface

http://www.worldofspectrum.org/forums/showthread.php?t=44508



Velesoft, der Hardwaremagier, entwickelt ein Universalinterface für den ZX Spectrum, der zu allem umgerüstet werden kann: K-Mouse, Joystick Flash EEPROM, SRAM, D/A Stereo Sound, Micro-SD Massenspeicher, etc.

ReRun

http://www.worldofspectrum.org/forums/showthread.php?t=42595&page=12

Toniman versucht ein OutRun Remake am Spectrum zu schreiben, mit dem Namen ReRun. Ein erstes Demo kann heruntergeladen werden. Es ist aber noch ein weiter Weg bis zum fertigen Spiel.

Ultimative Engine

http://www.boriel.com/forum/zx-basic-compiler/zxodus-engine-t891.html?sid=fcd993be279eeef301684aee08366b98

Cheveron hat die Zxodus][Engine angekündigt, allerdings auf Lizenzbasis für ausgewählte und wohlverdiente Coder. Lauffähig ist es auf dem Spectrum +2B, was den Nutzen stark einschränkt. Es soll folgende Features beinhalten:

- 9x9 multi-color (8x1 attribute) viewport
- 16 column text window (6x8 font)
- 256 multi-color (16x16 pixel) 2D tiles
- 256 multi-color (16x16 pixel) 3D dungeon tiles
- 128x128 tile world map
- 36x36 tile mini-map viewer
- 16 town maps of 32x32 tiles each
- 16 NPCs per town
- 11 character classes
- two spell systems
- melee and ranged combat
- Ultima 4 style dialog system
- AY music player
- BEEPer sound effects

Auf anderen 128K-Modellen wird es nicht perfekt funktionieren. Es gibt aber eine Umbaumöglichkeit für +2/128+ um das Timing anzupassen.

Limette Rick

http://www.worldofspectrum.org/forums/showthread.php?t=44932

Lime Rick ist ein neues Projekt von Einar Saukas, welches anfänglich für Bifrost* geplant wurde, jedoch schaut es nun so aus als ob eine völlig neue Grafikengine dafür verwendet werden soll.
Es wird die Umsetzung eines FLASH-Spiels, welches damit am Speccy besser als das Original aussieht.

Was ist los mit WOS?

http://www.worldofspectrum.org/infoseekid.cgi?id=0027992

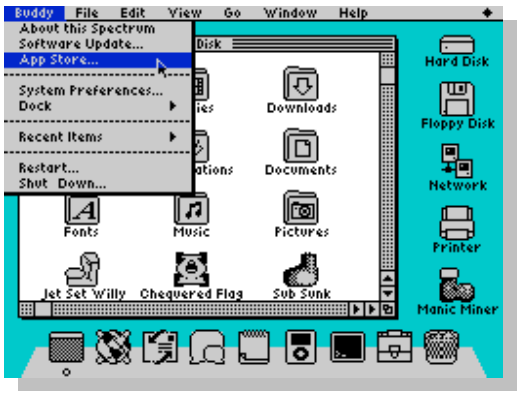
World of Spectrum hat seit Jahresmitte keine Updates mehr bekommen. Die Ursache dafür ist, dass Martijn einer Frau verfallen ist.

Ikonisch

http://www.worldofspectrum.org/forums/showthread.php?t=44356

IZX von Einar Saukas ist ein neues Standardformat für Icons am Spectrum.

Wer also für seine GUI-Projekte Ikons benötigt, der sollte sich den Treiber genauer anschauen.



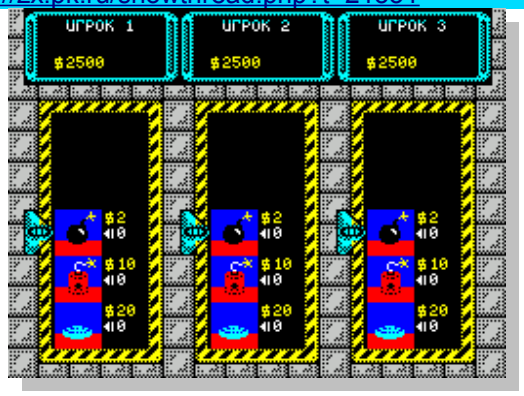
Scream Queen

http://www.worldofspectrum.org/forums/showthread.php?t=44643

Von utz stammt das Programm Zspeak, eine 2185 Bytes lange Routine die mit dem BEEPer einen Sprachsynthesizer nachbildet. Es ist auch auf 16K lauffähig.

Minen!!!

http://zx.pk.ru/showthread.php?t=21531

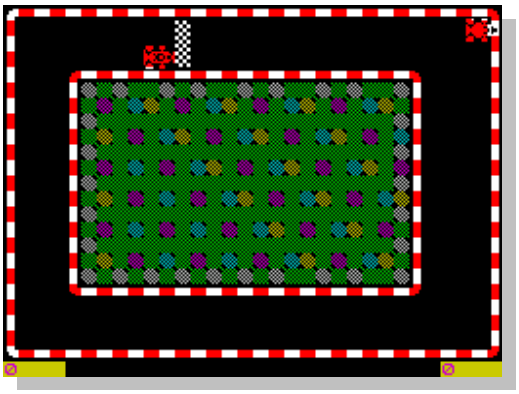


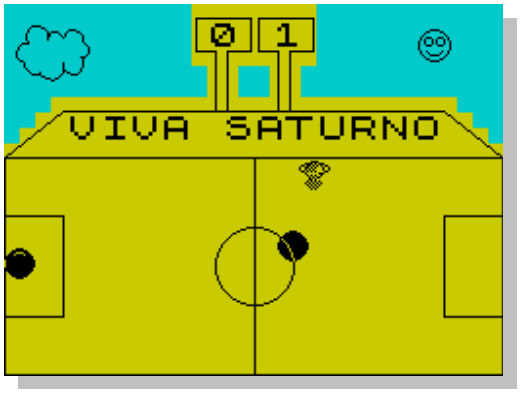
Minebomber ist die Umsetzung eines russischen Puzzle-Spiels vom PC. Leider gibt es das nur als SCL, und noch dazu auf russisch. Und als ob das nicht genug wäre, kann man es aufgrund von Floating Bus nur spielen, wenn ein Kempston Interface angeschlossen ist.

Coches & Ciclopes (y Saturno)

http://www.boriel.com/forum/gallery/coches-ciclopes-y-saturno-t886.html

Dies sind zwei Spiele von Apenao, die in ZXBC geschrieben wurden. Pixelpräzise Spritebewegung ist sicher eine Erwähnung wert. Und aus dem Sourcecode kann man etwas lernen.





Eigentlich wollte Apenao diese Spiele nicht freigeben und hat sie nur für seinen Sohn geschrieben, der ein Fan des Planeten Saturn ist.

Wumpusjagd geht weiter

„Hunt the Wumpus“ wird demnächst in einer aufpolierten Version erscheinen. Ich habe auch ein paar Grafikroutinen gehörig beschleunigt, und den Hänger bei der Pro-Maze Generierung beseitigt.

DivMMC EnJOY! für den Sinclair ZX Spectrum

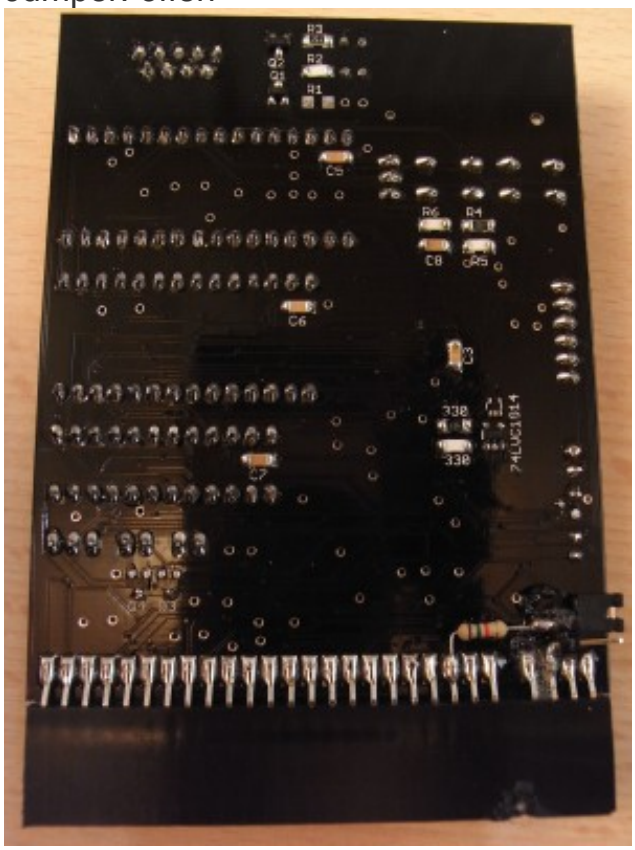
DivMMC EnJOY!

Das DivMMC EnJOY! ist der Nachfolger des in mehreren Variationen produzierten DivIDE. Ein kleiner Nachteil des DivIDE war das verwendete CF-Interface. Die CF-Karten werden etwas seltener und die SD-Karten sind sehr weit verbreitet und in vielen Größen erhältlich. Da auch fast jeder PC oder Laptop einen SD-Kartenslot hat, ist die Übertragung von Software hier besonders einfach.

Das DivMMC EnJOY! hat ein integriertes Kempston Joystick Interface für Atari kompatible Joysticks. Es gibt einen NMI-Knopf um in das Menü des Betriebssystems zu gelangen und einen Reset-Knopf. Es hat 128KB Speicher (das vierfache des DivIDE) und kann für zukünftige Funktionen einfach auf 512 KB Speicher erweitert werden. Das DivMMC wird mit ESXDOS ausgeliefert, welches mit FAT und FAT32 zusammenarbeitet und wodurch keine zusätzliche Software am PC benötigt wird. Der meist benutzte Dateityp bei ESXDOS dürfte „.TAP“ sein welcher in einer Datei ein komplettes Spiel enthält. Andere unterstützte Dateiformate sind „.SNA“ und Z80 Snapshot Dateien sowie „.TRD“ Dateien.
Natürlich kann man auch eigene Dateien auf die Karten speichern und davon laden.
Das DivMMC ist zu allen Sinclair Spectrum Modellen kompatibel.

Ben Versteeg hat wieder viel Arbeit in das Gerät gesteckt, wenn man die Beiträge im WOS-Forum im Laufe des Jahre 2013 etwas verfolgt hat. Es gab einige Klippen zu umschiffen um das Original-Design, das schon ein paar Jahre in der Schublade liegt und von Mario Prato mitentwickelt wurde, für eine richtige Produktion anzupassen. Die erste Auflage des Interfaces hat noch ein paar "Schönheitsfehler", die aber in der zweiten Auflage korrigiert werden. Zum einen wäre das ein zusätzlicher Jumper, der für den Betrieb an einem Spectrum +128 (Toastrack) nötig ist - zum anderen die falsche Beschriftung des Jumpers für den Joystick Anschluss. Außerdem ist es geplant für die zweite Auflage ein Gehäuse für das Interface anzubieten.
Die Platine ist wieder im typischen "Sinclair-schwarz" gehalten und kam zusammen mit einer SD-Karte, auf der schon einige Spiele und Demos sind und daher sofort betriebsbereit zum

Testen ist.
Die Einstellung der Jumper ist wie folgt:
ZX Spectrum 16K, 48K, 48+ oder +2 (graues Modell):
+2a/+3 Jumper: offen - +128 Jumper: offen
ZX Spectrum 128K "Toastrack":
+2a/+3 Jumper: offen - +128 Jumper: geschlossen
ZX Spectrum +2A, +2B oder +3:
+2A/+3 Jumper: geschlossen - +128 Jumper: offen



DivMMC EnJOY! - Rückseite - unten rechts der Jumper für den +128K Toastrack

Joystick Interface Einstellungen:
Um das DivMMC mit anderen Interfaces kombinieren zu können die ebenfalls einen Kempston Joystick Port haben, kann man den Joystickport am DivMMC deaktivieren.
Die Beschriftung in der ersten Auflage ist allerdings falsch - eigentlich sollte es "JOY1/0" heißen und nicht "JOY0/1". Um den Joystickport zu aktivieren einfach den Jumper in die linke Position setzen - um das Interface zu deaktivieren in die rechte Position setzen.
Um die Firmware des DivMMC - das ESXDOS auf eine neue Version upgraden zu können, muss der Jumper "Flash" geöffnet werden wenn die Software zum Upgrade dazu die Anweisung gibt. Das EEPROM ist schreibgeschützt, so dass ein unabsichtliches Programmieren nicht möglich ist. Sollte ein Firmware Upgrade schief gehen und anzeigen, dass das ROM geschützt ist, muss man das Utility herunterladen um den Schutz aufzuheben.

DivMMC EnJOY! am Spectrum 48K mit Joystick

Fazit
Eigentlich das Gleiche wie das DivIDE und doch nicht so ganz. SD-Karten sind schon praktischer zu verwenden - sonst gleichen sich die beiden Geräte. Genial ist natürlich der Kempston Joystickport. SD-Karte rein, Joystick dran, starten, spielen. Hier muss man nicht noch ein weiteres Interface anstecken um dann komfortabel mit dem Joystick spielen zu können - das ist schon sehr praktisch. Leider ist der Platzbedarf nach hinten

doch etwas größer (11,5cm) als z.B. beim DivIDE 2k11, das ja extrem klein ist und am Expansion Port senkrecht angeschlossen wird - auch wenn hier noch ein Joystick Interface angeschlossen wird, sind es nicht mehr als 8cm.
Bei diesem Interface entfällt die Wahlmöglichkeit zwischen mehreren Systemen - hatte man früher noch die Qual der Wahl zwischen FatWare und anderen Systemen, funktioniert hier nur noch das ESXDOS.
Außerdem bin ich immer etwas vorsichtig, wenn ich den NMI-Knopf drücke, da dabei das Interface leicht nach unten gedrückt wird und das evt. nicht so gut ist für die alte Hardware. Bei einem Test funktionierten Micro-SD-Cards im SD-Card Adapter ohne Probleme.
Das DivMMC EnJOY! hat die Lücke zur Funktionsvielfalt des Interface 1bis etwas kleiner gemacht :-)

Das Interface kostet 50 € zzgl. 10 € für den Versand – Kontakt:
ben.versteeg@benophetinternet.nl

Jungsi

BIFROST* für Grafiker, Teil 1

Von Einar Saukas

Die Nutzung von Multicolor erfordert zwei sehr unterschiedliche Fähigkeiten: *Programmierung* und *Grafik-Design.*

Ich habe schon mehrere Tutorials über diese Aufgabe geschrieben (auch die BIFROST* offizielle Dokumentation bietet eine Vielzahl von Informationen dazu), jetzt werde ich auf Letzteres fokussieren. Dieses Mal werde ich es vermeiden, zu technisch zu sein, so dass selbst wenn Sie ein Grafiker ohne Programmierkenntnisse (vielleicht in einem Team arbeitend, um ein neues Spiel in Multicolor zu entwickeln) sind, Sie in der Lage sein sollten, alles, was ich hier geschrieben habe, zu verstehen.

ÜBER BIFROST*

Ich bin sicher, dass Sie bereits wissen, dass der ZX-Spectrum nur 2 Farben (Ink und Paper) pro Zeichen (8x8 Pixel) unterstützt. Ein typischer Held in einem Spectrum Spiel sieht wie folgt aus:

** Grafik von Baron Ashler (von der ursprünglichen Knights & Demons -Version)*

Das obige Bild besteht aus 4 UDG Zeichen, die jeweils eine einzige Tintenfarbe (mit PAPER 0) haben:

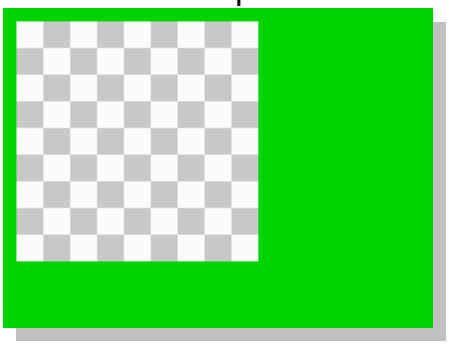
Mit BIFROST* ist es möglich, verschiedene INK / PAPER Farben pro Pixelzeile zu verwenden, so könnte man das Bild über das Hinzufügen weiterer Details, wie verschiedene Farben für Haare, Gesicht, Brust, Körperpanzer, usw. verbessern. Als Bonus unterstützt BIFROST* animierte mehrfarbige Bilder (in der Regel 4 Bilder), so dass die früheren Helden mit mehr interessanten Variationen verbessert werden, zum Beispiel in der Herstellung eines netten "marschieren" Ritters:

**Grafik von Baron Ashler (von "Knights & Demons DX"-Beta-Version)*

Habe ich jetzt Ihre Aufmerksamkeit erregt? Gut! Aber halt! Zuerst müssen wir ganz von vorne beginnen...

SCREEN REGIONEN

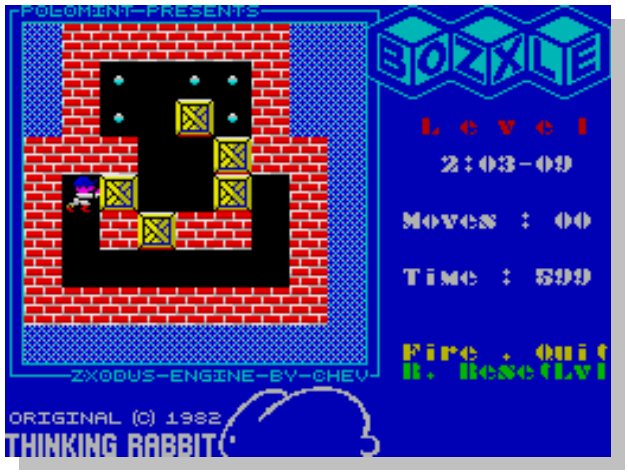
Aus technischen Gründen ist es nicht möglich, Multicolor auf dem gesamten Spectrum Bildschirm zu nutzen. In BIFROST*, hat die Multicolor-Region ein 9x9-Gitter mit jeweils 16x16 Pixeln, also einer Fläche von 144x144 Pixeln, die etwa die Hälfte des Bildschirms einnimmt. Alles, was in diesem Bereich (wie weiß im Bild unten markiert) ist, wird durch BIFROST* kontrolliert und zu Multicolor umgewandelt, alles, was außerhalb dieses Bereichs (grün im Bild unten) ist,funktioniert immer noch genau wie ein normaler Spectrum Bildschirm:



Diese Trennung ist deutlich sichtbar in der offiziellen BIFROST* Demo, die nur die Multicolor Region verwendet und nicht wirklich den Rest des Bildschirms:

** Grafik von R-Tape (von der offiziellen BIFROST* demo)*

Jedoch muss ein vollständiges Multicolor Spiel vorsichtiger mit der grafischen Darstellung sein, was ganz offensichtlich ist. Diese Spiele sollten in der Regel die Multicolor-Region für die wichtigste "Spielfläche" nutzen, und der Rest des Bildschirms kann ein schickes Hintergrundbild präsentieren welches Zusatzinformationen, wie aktuelle Level, Pukte, Timer, etc. Hier haben wir ein

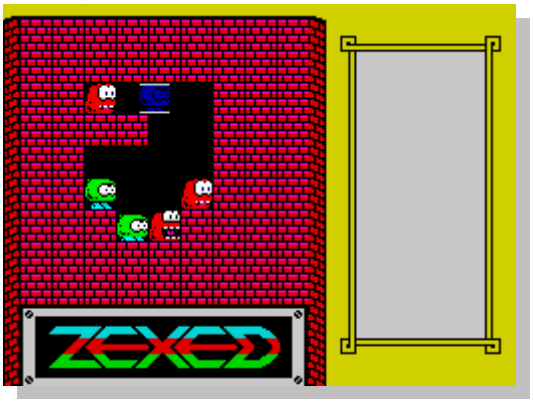


paar Beispiele:

** Grafik durch Polomint (von Bozxle welches ZXodus statt BIFROST* verwendet, aber in einer ähnlichen Art und Weise arbeitet)*

** Grafik von Einar Saukas (aus "MIDNIGHT RIDERS FANCLUB" Beta-*

Version)



* *Multicolor-Grafik von R-Tape, Hintergrundgrafiken von Einar Saukas (von "ZEXED" Beta-Version)*

In diesem letzten Beispiel wurde das Hintergrundbild absichtlich so entworfen, um die Trennung zwischen Multicolor und normalen Regionen zu verschleiern. Wenn Sie genau hinschauen, sollten Sie die vorderen Ziegel, die "direkt vom Sonnenlicht getroffen" sind, (eigentlich in der Multicolor Region) bemerkt haben. Die Farben Rot, Magenta und Schwarz sind da vorhanden. Die Steine an den Seiten die etwas "in den Schatten" sind (befinden sich in der regulären Region außerhalb von Multicolor) haben nur rote und schwarze Farben. Dieser Trick macht den Eindruck, dass die Spielfläche größer wirkt (jenseits der Grenze der Multicolor Region), so schlage ich ähnliche Lösungen vor, wenn möglich (natürlich ist es vom Spiel abhängig).

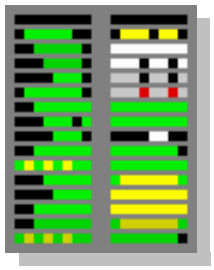
Aber wie kann man etwas in der Multicolor-Region zeichnen? Die Antwort ist einfach: *Multicolor Fliesen.*

MULTICOLOR KACHELN

Eine mehrfarbige Fliese ist ähnlich wie ein UDG Zeichen auf dem ZX-Spectrum, außer dass es größer (16x16 Pixel statt 8x8) und bunter ist.
Hier ist ein Beispiel mehrfarbiger Fliesen:

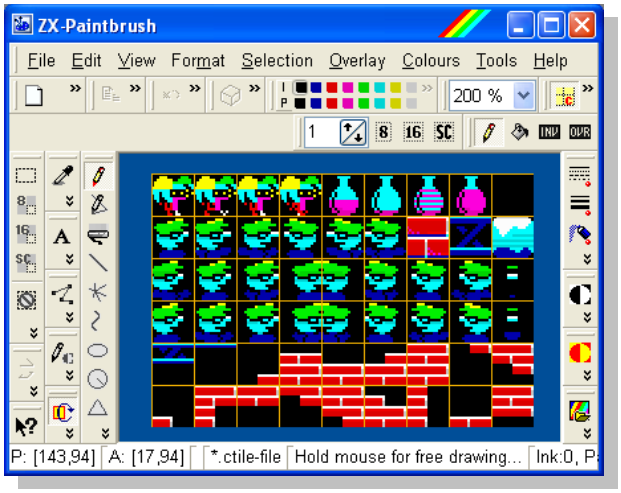
* *Grafik von R-Tape*

Schauen Sie genau hin, und Sie werden feststellen, dass jede "Linie" von 8 Pixel eine andere Kombination von zwei Farben (INK / PAPER) hat:



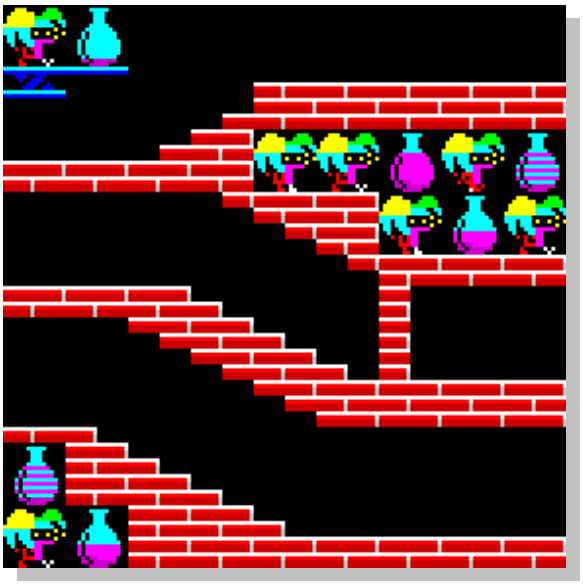
Alles, was Sie im Inneren der Multicolor-Region in einem BIFROST* Spiel sehen, wird mit mehrfarbigen Fliesen angezeigt, darunter animierte Monster und jedes Stück Landschaft. Eigentlich wäre es möglich, auch andere Verfahren zu verwenden, was aber in der Praxis nur selten vorkommen sollte, so dass wir diese andere Möglichkeiten derzeit besser ignorieren.

Zum Beispiel Demo6 verwendet die folgenden Multicolor Fliesen:



** Erste Reihe von mehrfarbigen Fliesen von R-Tape, zweite Reihe von mehrfarbigen Fliesen von na_th_an*

Mit diesen Multicolor Fliesen, ist es möglich, das folgende Szenario in der Multicolor Region zu machen:



Nun, wenn Sie das Bild oben sorgfältig betrachten, sollten Sie feststellen, dass der mehrfarbige Bereich oberhalb 9x9 Kacheln enthält, die genau mit dem 9x9-Gitter in der bereits erwähnten Multicolor Region korrespondieren. Es ist in der Tat nicht wirklich notwendig, jeden Multicolor Ziegel genau in eine Raster Position zu bringen. Die Position eines mehrfarbigen Kachels kann nach oben / unten durch eine beliebige Anzahl von Bildpunkten frei geändert werden, und links / rechts, um ein Vielfaches von 8 Pixeln (siehe demo3 um zu verstehen, was ich meine). Allerdings hat es Vorteile mehrfarbige Fliesen auf genaue Startpositionen zu legen:

- Multicolor Fliesen werden automatisch in BIFROST* animiert, nur wenn sie bei exakten Gitterpositionen angeordnet sind. In diesem Fall gibt es kein Limit, wie viele Fliesen animiert sind. Zum Beispiel in der multicolor Version von "Knights & Demons DX", werden alle 9x9 = 81 mehrfarbige Fliesen automatisch gleichzeitig animiert.
- Da auch die Landschaft mit mehrfarbigen Fliesen gebaut wird, hat es Sinn, dass Kacheln auf genaue Gitterpositionen platziert werden. Das macht es viel einfacher, dass alle Teile zusammen passen.
- Für Programmierer, die Manipulation der Fliesen auf genauen Gitterpositionen ist in der Regel viel einfacher und effizienter als auf den anderen Positionen.

Aus diesem Grund ist es wichtig das Spiel so zu gestalten, dass die Fliesen im Spiel möglichst exakt in die Gitterpositionen passen, aber es vermeiden, dies zu offensichtlich für den Spieler zu machen. Zum Beispiel das Plattform-Spiel, vorgestellt im Demo6 setzt alle Tiles auf genaue Gitter-Plätze, außer dem wandernden Helden und Feinden. Hoffentlich ist es nicht leicht das anhand von der Bildschirmfotos zu bemerken...

In "ZEXED", werden sich alle Kacheln (auch Monster) die meiste Zeit auf genauen Gitterpositionen befinden. Wenn ein paar Monster fallen, werden sie "von Hand" durch das Spiel vorübergehend animiert, aber sie stoppen immer auf genauen Gitterpositionen, so dass sie danach immer wieder automatisch durch BIFROST* animiert werden. Auch Buzzsaw+ (welches eine speziell

programmierte Render-Routine nutzt statt einer Multicolor-Engine) verhält sich in ähnlicher Weise.

Die einzige Ausnahme von dieser Regel ist "MIDNIGHT RIDERS FANCLUB", bei dem in bestimmten Levels die mehrfarbigen Fliesen 8 Pixel aus dem Gitternetz (horizontal und vertikal) herausragen. Dies war notwendig, um das Bild immer zentriert zu halten. Doch dieses Spiel nutzt keine animierten Tiles und es benötigt nicht viel Leistung, so war es kein Problem in diesem Fall. Sie müssen immer Vor- und Nachteile in solchen Fällen ausgleichen...

FAZIT

Jetzt sollten Sie ein gutes Verständnis darüber haben, wie Multicolor in der Praxis funktioniert. Im nächsten Teil dieses Tutorials werde ich anspruchsvollere Anwendungen von mehrfarbigen Fliesen erklären, und einige Ratschläge über die Gestaltung von Multicolor-Spielen bieten.

Wenn Sie in der Zwischenzeit versuchen wollen, neue Multicolor Fliesen jetzt schon mit ZX-Paintbrush zu erstellen, gehen Sie auf "Datei", "Neu" und wählen Sie "Bifrost* ZXodus ctile Dateien" (um eine einzige mehrfarbige Tile zu bearbeiten), oder erstellen Sie einen neuen Spectrum-Screen und wählen Sie "Extras", "Konvertieren Spektrum zu Timex-Format" (um einen ganzen Bildschirm mit mehrfarbigen Fliesen zu bearbeiten).

Eine weitere gute Möglichkeit zu lernen, wie man gute Multicolor Fliesen erstellen kann, ist der Import der Arbeit von jemand anderem in ZX-Paintbrush, um sie genauer zu untersuchen. Ich habe bereits erklärt, wie es zu tun ist im nachfolgenden Tutorial.

Viel Spaß!

Tutorial: ZX-Paintbrush and BIFROST*

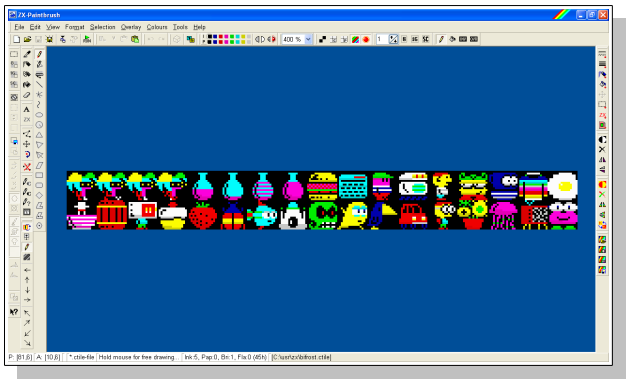
Einar Saukas
Übersetzung: LCD

Die neueste Version von ZX-Paintbrush unterstützt nun die Bearbeitung mehrfarbiger Fliesen wie sie in BIFROST*, ZXodus und ColorTILE (arbeitet nur auf dem Timex 2068) verwendet werden. In diesem Beitrag werde ich die einfachste Methode erklären, um es mit BIFROST* in der Praxis anzuwenden.

Der erste Schritt ist, eine binäre Datei mit den Kacheln zu erstellen, die Sie bearbeiten möchten. Verwenden Sie Ihren Lieblings-Emulator, laden Sie das BIFROST* Demo, dann wählen Sie "speichern als Binär-Datei" (oder "Speichern als Speicherblock" in einigen Emulatoren). Geben Sie einen Dateinamen mit der Endung ".ctile", wählen Sie Startadresse 48500 (oder 48500+n*64, wenn Sie die ersten n Fliesen überspringen möchten) und die Datenlänge 2176 (oder k*64, wenn Sie nur die Kacheln speichern wollen).

Nun doppelklicken Sie einfach auf die ".ctile" Datei, die Sie gerade im vorherigen Schritt gespeichert hatten. Das wird ZX-Paintbrush automatisch öffnen. Wenn Sie gefragt werden, ob Sie alle Kacheln zusammen bearbeiten

möchten, wählen Sie "Ja". Es wird wie folgt aussehen:

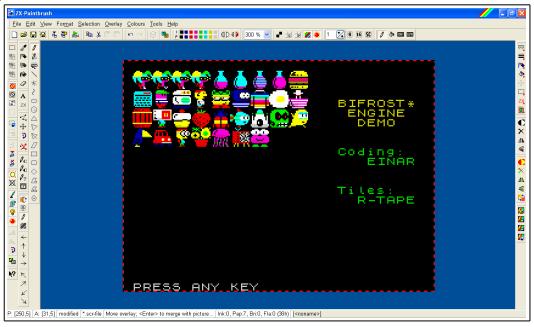


Jetzt können Sie frei die Kachelbilder bearbeiten. Nachdem Sie die Bearbeitung abgeschlossen haben (und gespeichert, natürlich), gehen Sie zurück zu Ihrem Lieblings-Emulator und, während das BIFROST* Demo läuft, wählen Sie "Lade Binär-Datei" (oder "Lade Speicherblock" in Abhängigkeit von dem Emulator). Wählen Sie die Datei, die Sie (mit der Endung ".ctile") bearbeiten und geben Sie die gleiche Start-Adresse wie zuvor. Sie sollten bereits sehen wie Ihre bearbeiteten Bilder die Originale ersetzen!
Das gleiche Verfahren wird auch mit anderen Multicolor Kacheln-Programmen arbeiten, obwohl die Fliesen sich nicht immer bei der Adresse 48500 befinden.

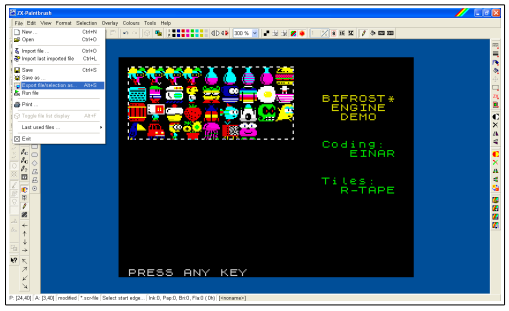
Eine andere Methode ist es, Fliesen direkt aus einem Programmfenster zu extrahieren, wie ich hier zu erklären versuche.

Verwenden Sie Ihren Lieblings-Emulator, speichern Sie einen Screenshot, der die Kacheln, die Sie bearbeiten möchten, zeigt. In diesem Fall speichern sie als BMP-oder PNG. JPG ist nicht geeignet da Kompressionsartefakte Bilddetails zerstören werden. Zum Beispiel wird das BIFROST* Demo dieses Bild produzieren:

Nun starten Sie ZX-Paintbrush und klicken auf das Symbol mit der Aufschrift "Convert Spectrum to Timex-Format". Das wird ZX-Paintbrush zum Timex-Modus umschalten. Danach wählen Sie "Datei importieren" aus dem Menü, wählen Sie Ihre BMP- oder PNG-Datei, und klicken Sie auf "OK". Es wird wie folgt aussehen:



Jetzt drücken Sie ENTER um den Import dieses Bild zu bestätigen, und Sie können danach frei diese Fliesen bearbeiten. Nachdem Sie die Bearbeitung beendet haben, ändern Sie "Rastergröße" zu 8 Pixel (klicken auf das Symbol "8" oben), Auswahlmodus aktivieren, wählen Sie die Multicolor Kacheln, und wählen Sie "Export Datei / Auswahl":



Vergessen Sie nicht, "Bifrost* ZXodus ctile Dateien" zu wählen, wenn Sie diese speichern. Nun, wenn Sie Ihre neue Reihe von Fliesen testen möchten, laden Sie das BIFROST* Demo mit Ihrem Lieblings-Emulator, wählen Sie dann "Lade Binär-Datei" (oder "Lade Speicherblock") bei der Adresse 48500. Schließlich, wenn Sie Ihre neuen Fliesen auf Band speichern, drücken Sie BREAK (die BASIC Version des BIFROST* Demos ermöglicht BREAK) und führen aus:
SAVE "Tiles" CODE 48500, k*64
um nur die ersten k Fliesen zu speichern.

Sprite Pack 1 für z88dk, Teil 2

Das Sprite-Pack besteht derzeit aus neun großen unabhängigen Modulen:

- SPRITES: Unterstützt die Erstellung, Bewegung und Löschung einer beliebigen Anzahl von Sprites auf dem Bildschirm. Sprites können pixelgenau dank einem integrierten Software-rotator platziert werden und man kann eine von vier Arten verwenden: Maske, OR, XOR und Ladung. Hintergrund Tiles decken den Bildschirm-Hintergrund, wenn Sprites verwendet werden.
- COMPLETE IM2 INTERRUPT API:
  Interrupt-Service-Routinen können installiert und auf jedem Vektor deinstalliert werden. Eine speziell entwickelte universelle Interrupt-Service-Routine kann auf einem beliebigen Vektor installiert werden, und kann eine beliebige Anzahl von Hooks mit ihr verbunden haben. Auf Interrupt wird jeder dieser Hooks der Reihe nach aufgerufen.
- INPUT: Lesen von der Tastatur, Joysticks und Mäusen.
- SCREEN-ADRESSE HELFER:
  Eine Reihe von Funktionen, die Bildschirm-Adressen von Pixel- oder Zeichen-Koordinaten berechnen kann. Funktionen können auch die Bildschirmadresse nach oben, unten, links oder rechts, um Pixel- oder Zeichenmengen bewegen.
- RECHTECK KREUZUNGEN :
  Unterprogramme, die Kollisionen zwischen Rechtecke, Intervallen und Punkten erkennen kann. Perfekt für Kollisionserkennung von Sprites oder Überwachung einer "Hot Spot" für einen grafischen Zeiger.
- DYNAMIC BLOCK SMEMORY ALLOCATOR:
  Eine schnelle, kompakte Blockspeicherzuweisung. Ergänzt eingebauten malloc C-Funktionen.
- Abstrakte Datentypen: Teilgebiete umfassen verketteten Liste, Hash-Tabelle und Heap-Datentypen.
- Datenkompression: Momentan besteht diese aus einem statischen Huffman Kompressor / Dekompressor.
- SONSTIGES: Beinhaltet einen Zufallszahlengenerator, Multiplikationen, Endian-Swap, einer Muster Flutfüllung usw.

Dieses Tutorial beschreibt die C-API und nimmt an, dass der Spectrum Video-Modus vor der Bibliothek Zusammenstellung ausgewählt wurde. Es wird hilfreich sein, eine Kopie von "spritepack.h" geöffnet zu haben,

während Sie dies lesen. Diese Datei enthält alle Funktionsprototypen von SP. Suchen Sie nach der Funktion von Interesse und Sie werden sehen, welche Parameter erwartet werden, wenn die Funktion aufgerufen wird. Sie werden auch feststellen, dass alle Datenstrukturnamen und Funktionsnamen, die in der SP-Bibliothek vorhanden sind, ein "sp_“ vorangestellt haben. Dies sollte besonders für neue C-Programmierer hilfreich sein, so dass sie leicht erkennen, was ein Teil der C-Sprache ist und was nicht.

SPRITES & Hintergrund Kacheln

Das Sprite-Modul ist um einen Differenzbild-Updater, der die Anzeige in das vertraute 32x24 Array von Zeichenzellen teilt, aufgebaut. Jede Charakter-Zelle enthält genau eine Hintergrund Kachel und kann eine beliebige Anzahl von Sprite-Zeichen enthalten. Der Updater ist „differenziell", weil es nicht wirklich den ganzen Bildschirm neu zeichnet, wenn ein Update auftritt. Es wird nur diese Charakter-Zellen neu zeichnen, die sich seit der letzten Aktualisierung geändert haben. Informationen, welche der Zeichenzellen modifiziert wurden ist in einer internen Datenstruktur gespeichert. Die Kennzeichnung spezifischen Charakter Zellen welche in der nächsten Aktualisierung neu gezeichnet werden, wird als "Entwertung" bezeichnet, und die Markierung, welche keine Aktualisierung benötigt, heißt "Validierung". Die meisten Funktionen, die die SP Hintergrund Kacheln oder Sprites verändern „entwerten“ automatisch die Zellen, die sie beeinflussen, so dass der Bildschirm richtig in der nächsten Aktualisierung neu gezeichnet wird. Sie können mehr Kontrolle darüber, welche Bereiche auf dem Bildschirm neu gezeichnet wird, oder nicht, durch Aufrufe von "sp_Invalidate" und "sp_Validate" erlangen.

Die beiden grafischen Einheiten, die das Sprite-Modul versteht, sind die Hintergrund Kachel und das Sprite. "Hintergrund Kachel" ist in Wirklichkeit ein ausgefallener Name für einen farbigen UDG. Sie werden in fast der gleichen Weise wie die Buchstaben und echte UDGs auf dem Bildschirm in BASIC gedruckt. Hier ist ein Beispiel, das den Buchstaben "A" in schwarz auf weiß an Position (10,12) auf dem Bildschirm zeichnet:

```
sp_PrintAtInv(10,12,    INK_BLACK    |
PAPER_WHITE, 'A');
```

Es gibt mehrere Varianten dieser Funktion (siehe sp_PrintAt, sp_PrintAtInv, sp_PrintAtDiff), die sich nur darin unterscheiden, ob sie die Charakterzelle beim Zeichnen ungültig markieren oder nicht. In diesem Beispiel wird "sp_PrintAtInv" verwendet, das neben dem Drucken des Buchstabens "A", auch die Zelle ungültig markiert, so dass der Buchstabe in der nächsten Bildschirmaktualisierung gezeichnet wird. ERINNERUNG: Keiner der Befehle der SP ändert sofort das Display! Die Anzeige wechselt auf einmal erst, wenn der Bildschirm in einem Aufruf von "sp_UpdateNow" aktualisiert wird.

Zeit für ein Beispiel. Da wir noch nicht sehr weit gekommen sind, werde ich Sie

noch kurz einführen in eine Handvoll von SP Funktionen, die noch nicht diskutiert wurden. Sie werden später ausführlicher erörtert. Eine kompilierte TAP Datei ist ebenfalls verfügbar:

(http://www.timexsinclair.org/alvin/spritepack/AB.zip)

Sie werden zwei Nicht-Standard-Elemente in diesem Listing bemerkt haben, wenn Sie irgendeine Art von Hintergrund in der Programmiersprache C haben. Die spezielle "pragma" Direktive erzählt z88dk, dass wir das Programm mit Stack Pointer (SP) des Z80 auf 61440 gesetzt starten wollen. Spectrum BASIC legt den Stapel auf einer ungünstigen Stelle, wir verschieben ihn, damit er sich unter einigen Tabellen, die automatisch von SP initialisiert werden wenn "sp_Initialize ()" ausgeführt wird, befindet. Fügen Sie diese Direktive in allen SP-Programme kurz vor main()-Funktion von C, ein. Sps Memory map wird genauer beschrieben

```
#include <stdlib.h>
#include <spritepack.h>
extern struct sp_Rect *sp_ClipStruct;
#asm
LIB SPCClipStruct
._sp_ClipStruct         defw SPCClipStruct
#endasm
#pragma output STACKPTR=61440
main()
{
   uint keyI, keyC;
   #asm
   di
   #endasm
   sp_InitIM2(0xf1f1);
   sp_CreateGenericISR(0xf1f1);
   #asm
   ei
   #endasm
   sp_Initialize(INK_BLACK | PAPER_WHITE, ' ');
   keyI = sp_LookupKey('i');
   keyC = sp_LookupKey('c');
   while(1) {
      sp_UpdateNow();
      sp_PrintAt(rand()%24, rand()%32, INK_YELLOW | PAPER_BLUE, 'A');
      sp_PrintAtInv(rand()%24, rand()%32, INK_RED | PAPER_CYAN, 'B');
      sp_Pause(20);
      if (sp_KeyPressed(keyC)) {
         sp_ClearRect(sp_ClipStruct, INK_BLACK | PAPER_WHITE, ' ',
sp_CR_TILES);
         sp_Invalidate(sp_ClipStruct, sp_ClipStruct);
      } else if (sp_KeyPressed(keyI))
         sp_Invalidate(sp_ClipStruct, sp_ClipStruct);
   }
}
```

in dem Abschnitt über dynamische Blockspeicherzuweisung.

Das andere Ungewöhnliche sind die eingebetteten Assemblerbefehle "di" und "ei", die Interrupts deaktivieren, während der eingeschlossene C-Code ausgeführt wird. Für dieses Programm muss die BASIC Interrupt-Service-Routine deaktiviert sein, weil eine der aufgerufenen Funktionen den IY Register nutzt (sp_Invalidate, siehe auch „Liste aller Funktionen in SP mit IY“). Die Interrupt-Service-Routine von Spectrum BASIC erwartet, dass der IY Register niemals verändert wird. Ich konnte einfach Interrupts mit "#asm; di; #endasm" deaktivieren, aber dann würde die Funktion "sp_Pause" sich sperren, die die Interrupts zählt um Zeit zu messen. Die Alternative, die ich hier verwendet habe, ist meine eigene Interrupt-Service-Routine (die nichts tut) zu schreiben, um die von Basic zu ersetzen. Es ist nicht notwendig für Sie, dies jetzt schon zu verstehen, denn das wird alles vollständig in dem Abschnitt „INTERRUPTS“ erklärt werden.

"Sp_ClipStruct" ist eine der Sprite Pack Variablen. Seine seltsam aussehende Erklärung wurde von der SP-Header-Datei „spritepack.h“ kopiert. Es gibt etwa ein Dutzend dieser Variablen in der Header-Datei, die in Ihrem Programm verwendet werden können, durch Deklaration in der Hauptdatei auf diese Weise. "Sp_ClipStruct" ist ein Zeiger auf ein Rechteck, dass den Umriss des ganzen Bildschirms in Zeichen definiert. Mehr Informationen darüber, warum diese Erklärung so ist, wie sie ist, kann im Sprite-Pack-Abschnitt über Variablen gefunden werden.

"Sp_Initialize" initialisiert das Sprite-Modul und muss aufgerufen werden, bevor Sprites oder Hintergrundkacheln verwendet werden. In diesem Fall führen die Parameter dazu, dass der Bildschirm gelöscht wird mit schwarz auf weiß Leerzeichen.

"Sp_LookupKey" liefert einen Scan-Code für ein ASCII-Zeichen, das die Tastendrücke darstellt, notwendig, um das Zeichen zu erzeugen. Ein späterer Aufruf von "sp_KeyPressed" übernimmt diesen Scan-Code und gibt true zurück, wenn die Taste zur Zeit gedrückt wird.

"Sp_Pause"-Funktionen genau wie der Basic PAUSE-Befehl.

Der "sp_Invalidate" Aufruf akzeptiert zwei Rechtecke in Char-Einheiten. Der Schnittpunkt dieser beiden Rechtecke definiert den Bereich des Bildschirms, der für die Aktualisierung markiert wird. Typischerweise nutzt das zweite Rechteck den kompletten Bildschirm (dh "sp_ClipStruct") und das erste Rechteck definiert den Bereich, der als ungültig markiert werden sollte, der über die Grenzen des Bildschirms reichen kann. In diesem Beispiel wird "sp_ClipStruct" für beide Rechtecke verwendet, wodurch der ganze Bildschirm für ungültig erklärt wird.

"Sp_ClearRect" tut, was der Name vermuten lässt - es löscht eine rechteckige Fläche auf dem Bildschirm. In diesem Fall wird der Vollbild gelöscht ("sp_ClipStruct") mittels schwarz auf weiß Leerzeichen. Das Flag "sp_CR_TILES" bedeutet, dass nur Hintergrundkacheln gelöscht werden. Andere Flag Einstellungen sind

"sp_CR_SPRITES" – um nur Sprites zu löschen - und "sp_CR_ALL" Sprites und Kacheln zu löschen. Diese Funktion markiert *nicht* den Bereich den es löscht, was bedeutet, dass der Bereich nicht unbedingt in der nächsten Bildschirmaktualisierung neu gezeichnet werden wird. Der folgende Aufruf "sp_Invalidate()" stellt sicher, dass der Bereich neu gezeichnet wird.

Bei jedem Durchlauf der Hauptschleife werden zwei Buchstaben an zufälligen Stellen auf dem Bildschirm ausgegeben. As sind ohne Entwertung, und Bs sind mit Entwertung gedruckt. Jedes Mal beim Schleifendurchlauf wird ein Aufruf an "sp_UpdateNow" gemacht, um Teile des Bildschirms, die für ungültig erklärt worden sind, neu zu zeichnen. Dies bewirkt, dass alle "B"s zu sehen sind, aber keine "A"s! Die "A"s *werden* gedruckt, sind aber nicht für ungültig erklärt, so dass sie nicht angezeigt werden, wenn der Bildschirm aktualisiert wird. Um dies zu überprüfen, halten Sie die "i"-Taste auf der Tastatur gedrückt. Dies zwingt den gesamten Bildschirm für ungültig erklärt zu werden (und deshalb neu gezeichnet) und die "A"s erscheinen plötzlich. Durch Drücken von 'c' wird der Bildschirm gelöscht.

Eine weitere Funktion, die Sie nützlich finden könnten, ist "sp_ScreenStr()", die analog zu BASIC SCREEN$ und ATTR Funktionen arbeitet. Es gibt den Zeichencode und die Farbe der *Hintergrundfliese* zurück, die eine bestimmte Position auf dem Bildschirm einnehmen. Beachten Sie, dass Sprites die über den Hintergrund Kacheln schweben, beeinflussen können wie die Zeichenzelle tatsächlich auf dem Bildschirm erscheint. Wenn Sie direkt mit der Display-Datei umgehen wollen und dem was tatsächlich auf dem Bildschirm sichtbar ist, sollten Sie die SCREEN-ADRESSE -Funktionen untersuchen.

```
uint i;
i = sp_ScreenStr(10, 12);
```

In diesem Fall wird der Code der Hintergrund Kachel an Position (10,12) in den unteren 8 Bits der Variable "i" sein und die Kachelfarbe an der jeweiligen Position in den höherwertigeren-8-Bits von "i" sein.

"Sp_GetTiles()" speichert die Hintergrundkacheln in einem rechteckigen Bereich auf dem Bildschirm in ein Feld. "Sp_PutTiles()" druckt diese Hintergrundkacheln zurück auf den Bildschirm. Dies kann hilfreich sein, zum Beispiel, wenn Sie ein temporäres Menü wollen. Zunächst würden Sie mit "sp_GetTiles()" speichern, was auf dem Bildschirm unter dem Menü ist, dann würden Sie Ihr Menü (mit "sp_PrintString()", wahrscheinlich), Menübefehle bearbeiten, und schließlich das Menü durch die Wiederherstellung der Bildschirmbereiches darunter mit "sp_PutTiles()" löschen. Das zur Speicherung der Hintergrundbilder verwendete Array benötigt zwei Bytes pro Zeichen-Zelle im rechteckigen Bereich. Sie können auch "sp_GetTiles()" verwenden, um eine rechteckige Fläche aufzeichnen und dann mit "sp_PutTiles()" den Bereich auf dem Bildschirm an mehreren anderen

Positionen zu drucken, indem Sie das Speicher-Array als Makro benutzen. Eine noch bessere Möglichkeit, dies zu tun ist "sp_PrintString()" zu verwenden.

"Sp_PrintString()" gibt eine Zeichenfolge auf dem Bildschirm als Hintergrund Kacheln aus. Das Besondere ist, dass es die aktuelle Druckposition benutzt und der String eingebettete Befehle enthalten kann. Eine Auflistung der Byte-Befehle,um sie von dieser Funktion zu verstehen, finden Sie in der Tabelle 1.

Die 'N's in der Tabelle sind Platzhalter für Byte-Größe-Parameter und die 'W's sind Platzhalter für Wortformat-Parameter (2 Bytes). Zum Beispiel, bei Code 22 ("AT") lesen wir: es folgen zwei Bytes die neue y-Koordinate und dann die neue x-Koordinate halten.

Die String-Druck-Funktion druckt Text in einen Begrenzungsrechteck. Keine Zeichen auf den Bildschirm außerhalb des Rechtecks werden ausgedruckt. Wenn 'xwrapping' aktiviert ist, wird der Text umgebrochen, wenn der horizontale Rand des Rechtecks erreicht wird. Wenn 'yinc' aktiviert ist, wird dieser horizontale Umbruch auch die y-Koordinate erhöhen. Ohne xwrapping wird die Druckposition weiterhin außerhalb des Begrenzungsrechteck aktualisiert werden, aber Text wird nicht auf dem Bildschirm erscheinen, bis die Druckposition auf das Begrenzungsrechteck zurückkehrt. Alle Koordinaten sind relativ zur oberen linken Ecke des umschließenden Rechtecks.

Das Begrenzungsrechteck, aktuelle Druckposition, Farbe und verschiedene Mode-Bits werden in einer Print String Struktur gespeichert (struct sp_PSS).

Wenn die Zeichenfolge gedruckt wird, wird diese Struktur aktualisiert und wird daher ihren Status beim nächsten Druckfunktion-Aufruf beibehalten. Am einfachsten ist es, Änderungen an dieser Struktur durch Byte-Befehlscodes in einem gedruckten String machen, aber Sie können Änderungen direkt auf die Struktur machen, falls gewünscht. Die reservierten Felder in der Struktur müssen aktualisiert werden, wenn Sie Änderungen an den Grenzen des Rechtecks oder den x/y-Koordinaten machen. Sie können dies mit einem expliziten Aufruf von "sp_ComputePos" oder mit einem eingebetteten Befehl wie "AT" oder "Home" in der nächsten gedruckten Zeichenfolge tun.

```
struct sp_PSS {
   struct sp_Rect *bounds;  /* bounding rectangle for printed text */
   uchar flags;    /* bit 0=invalidate?, 1=xwrap?, 2=yinc?, 3=onscreen?
(res.) */
   uchar x;                 /* current x coordinate relative to bounding
rect  */
   uchar y;                 /* current y coordinate relative to bounding
rect  */
   uchar colour;            /* current attribute */
   void *dlist;             /* reserved */
   void *dirtychars;        /* reserved */
   uchar dirtybit;          /* reserved */
};
```

0	0x00	Terminate string	
5	0x05	NOP	Ignored character
6	0x06	Goto N	Goto X coordinate on the same line
7	0x07	Print String W	Print string at address W (like a subroutine call)
8	0x08	Move Left	
9	0x09	Move Right	
10	0x0a	Move Up	
11	0x0b	Move Down	
12	0x0c	Home	Move to top left corner of bounds rectangle
13	0x0d	Carriage Return	X coordinate to 0, Y coordinate increased
14	0x0e	Repeat N	Begins a block within string
15	0x0f	End Repeat	Ends the block, repeating it N times
16	0x10	Ink N (0-7)	
17	0x11	Paper N (0-7)	
18	0x12	Flash N (0 / 1)	
19	0x13	Bright N (0 / 1)	
20	0x14	Attribute N	Set colour to an 8-bit attribute value
21	0x15	Invalidate N (0 / 1)	Set Invalidation mode
22	0x16	At N (y), N (x)	
23	0x17	At N (dy), N (dx)	Relative AT. The print position will be moved a distance (dy,dx) characters
24	0x18	X Wrap N (0 / 1)	Enable or disable wrapping in X direction
25	0x19	Y Inc N (0 / 1)	Enable or disable increasing Y coordinate when X Wrap occurs
26	0x1a	Push State	Save current print state
27	0x1b	Escape	Next byte in string will be interpretted as a printable character instead of a command code
28	0x1c	Pop State	Pop current print state
29	0x1d	Transparent Character	If in invalidate mode, prints a "nothing" but marks the character cell for update. Will change the colour if attribute != 0x80

Stringbefehle

Lassen Sie uns mal ein Beispiel in Aktion sehen. Sehen Sie, ob Sie sich vorstellen können, was das folgende Programm tut und vergleichen dann die Ergebnisse, indem Sie die TAP Datei (http://www.timexsinclair.org/alvin/spritepack/PString.zip) in Ihrem Lieblings-Emulator starten.

```
#include <spritepack.h>
/* embedded relative AT commands would be more compact */

uchar *special = "\x16\x00\x00\x0e\x04\x1a\x0e\x05\x14\x16AB\x14\x38 \x0b" \
               "\x08\x08\x08\x14\x16CD\x14\x38 \x0a\x0f\x1c\x0b\x0b\x0b\x0f";

/* UDG Graphics */

uchar hash[] = {0x55,0xaa,0x55,0xaa,0x55,0xaa,0x55,0xaa};
uchar A[] = {0x55,0xaa,0x54,0xa9,0x01,0x7f,0x3e,0x8f};
uchar B[] = {0x95,0xaa,0x95,0xca,0xc0,0xfe,0xbc,0xf8};
uchar C[] = {0x45,0xa6,0x57,0xa7,0x4e,0xac,0x49,0x92};
uchar       D[] = {0xa1,0x6a,0xe5,0xf2,0x75,0x62,0x19,0x0a};

/* Initial State of Print String Struct */

struct sp_Rect r = {9,0,15,32};        /* Bound Rect @ (9,0) height=15,
width=32 */
struct sp_PSS ps = {
   0,                                  /* Illegal C to put '&r' directly here
*/
   sp_PSS_INVALIDATE | sp_PSS_XWRAP,   /* Invalidate Mode, X Wrap */
   0,                                  /* X = 0 */
   0,                                  /* Y = 0 */
   INK_YELLOW | PAPER_RED,             /* Attribute = yellow on red */
   0,                                  /* reserved */
   0,                                  /* reserved */
   0                                   /* reserved */
};

#pragma output STACKPTR=61440

main()
{
   uchar x;

   #asm
   di
   #endasm
   sp_InitIM2(0xf1f1);
   sp_CreateGenericISR(0xf1f1);
   #asm
   ei
   #endasm

   sp_TileArray(' ', hash);
   sp_TileArray('A', A);
   sp_TileArray('B', B);
   sp_TileArray('C', C);
   sp_TileArray('D', D);
   ps.bounds = &r;            /* since we couldn't do it in the static
initializer */
```

```
    sp_Border(YELLOW);
    sp_Initialize(INK_BLACK | PAPER_CYAN, ' ');
    for (x=15; 1; x=(x-1)&0x1f) {
        sp_UpdateNow();
        special[2] = x;
        sp_PrintString(&ps, special);
        sp_Pause(10);           /* hold down a key to see it go faster */
    }
}
```

Der "spezielle" String enthält Befehlscodes, die einen einzelnen Stern druckt (definiert durch Drucken der Buchstaben "AB" und dann "CD" direkt darunter) in einem 5x4 Array, 20-mal wiederholt. Unter dieser Deklaration sind Deklarationen für fünf UDG Grafiken. Diese werden in genau der gleichen Weise definiert, wie sie in Basic definiert werden, wo jedes Byte ein Bit-Muster für eine Zeile einer 8x8 UDG Grafik definiert.

Die Print String Struct wird als nächstes deklariert, mit einigen der Felder initialisiert, wie in den Kommentaren erklärt. Die reservierten Felder werden initialisiert, wenn der String gedruckt wird, wenn der Befehl "AT" als erster Punkt in der Zeichenfolge angetroffen wird (die Alternative wäre, "sp_ComputePos" explizit aufzurufen). Die Anfangskoordinaten sind relativ zu den Rechteckbegrenzungen. Da das Begrenzungsrechteck auf (Y,X) = (9,0 auf dem Bildschirm) liegt, ist die Ursrungskoordinate (0,0) tatsächlich bei (9+0,0+0) auf dem Bildschirm abgebildet. Der XWRAP Flag ist gesetzt, aber YINC nicht. Dies bewirkt, dass Zeichen die über den rechten Rand des Begrenzungsrechtecks gedruckt werden, wieder auf die erste Spalte des Begrenzungsrechtecks drüberschreiben. Da YINC nicht gesetzt ist, wird die Y-Koordinate nicht erhöht werden, wenn dieses Umkippen des Textinhaltes auftritt.

Weiter begegnen wir etwas Neuem. Bis jetzt werden Sie bemerkt haben, dass wir Hintergrundkacheln gedruckt haben, indem Sie einen Zeichencode (z. B. der Buchstabe "A") spezifiziert haben. Ich habe nicht erwähnt, wie diese Zeichencodes mit einer tatsächlichen grafischen UDG Darstellung verbunden sind. Dies wird durch das Kacheln-Array gemacht, welches ein Array von Zeigern auf UDG Grafikdefinitionen ist, eines für jeden Zeichencode. Sprite Pack initialisiert dieses Array automatisch indem es dieses in das ROM Fontset leitet, und das ist der Grund, warum wir fröhlich etwas auf dem Bildschirm ausgeben könnte, ohne uns irgendwelche Gedanken zu machen. Die "sp_TileArray" Aufrufe ändern die Grafikdefinitionen für bestimmte Zeichencodes auf etwas Neues. Das erste ändert die "Hash"-Grafik mit dem Space-Zeichen. Die nächsten paar Anrufe verbinden die Grafikzuweisungen der Buchstaben A bis D zu einem 2x2 -Stern.

Der "sp_Border" Aufruf setzt die Rahmenfarbe. Der "sp_Initialize" Aufruf löscht den Bildschirm mit einem Leerzeichen schwarz auf cyan. Das

Leerzeichen ist mit einer Hash-Grafik verbunden worden, daher der neue Hintergrund.

Das "x" Variable der FOR-Schleife wird verwendet, um die Position des Stern-Arrays zu verfolgen. Sein Wert wird direkt in die spezielle Zeichenfolge erster AT-Befehl, x-Koordinate gepoket. So ist die Bewegung erzielt worden.

SP definiert Etiketten für Farben, die OR-verknüpfbar sind, um einen numerischen Attributwert zu erzeugen. Die Liste :
INK_BLACK, INK_BLUE, INK_RED, INK_MAGENTA, INK_GREEN, INK_CYAN, INK_YELLOW, INK_WHITE, PAPER_BLACK, PAPER_BLUE, PAPER_RED, PAPER_MAGENTA, PAPER_GREEN, PAPER_CYAN, PAPER_YELLOW, PAPER_WHITE, BRIGHT, FLASH, TRANSPARENT.
Ein helles Attribut mit grüner Tinte und gelbem Papier könnten Sie (INK_GREEN|PAPER_YELLOW|BRIGHT) nutzen, anstatt die numerische Abbildung 116.
Der spezielle " TRANSPARENT "-Wert (128 - blinkt schwarz auf schwarz) ist reserviert, für die transparente Farbe: jede vorhandene Farbe wird nicht verändert. Diese spezielle Farbe wird nur beim Zeichnen von Sprites und Drucken des speziellen 'transparent char' mit "sp_PrintString()" unterstützt. Die Standard-Farben Schwarz, Blau, Rot, Magenta, Grün, Cyan, Gelb und Weiß sind auch für die Verwendung mit der Funktion "sp_Border" definiert. Sie sind in der Tat numerisch identisch mit den INK_ * Makros.

Die letzte Hintergrund Kachel-Funktion in der aktuellen Version der SP ist "sp_Pallette()". Diese Funktion ist nur im Timex Hi-Color- Modus anwendbar, mit der Farbauflösung 32x192 statt der 32x24 der Standard-Spectrum-Anzeige. Die Palette Array-Funktionen ist sehr ähnlich wie das Kachel-Array. Ein Farb Tile (Palette Eintrag) von 0-255 ist mit einem 8-Byte-'Attribut' UDG verbunden. Der Attributwert in Funktionsaufrufen ist dieser Single-Byte- Palette Eintrag, der als look up für die 8 Farben in jeder Zeichen Zelle verwendet wird. Standardmäßig werden alle Palette-Einträge auf eine solide schwarz auf weiß Farbe zeigen, wenn hi- color-modus benutzt wird.

Das andere grafische Objekt, das vom Sprite-Modul verstanden wird, ist das Sprite. Sprites sind grafische Objekte, die über dem Hintergrund und untereinander schweben. SP kann eine beliebige Anzahl von Sprites in jeder Größe unterstützen, mit offensichtlichen Leistungseinbußen, wenn die Anzahl und Größe der *bewegten* Sprites auf dem Bildschirm ansteigt (man erinnere sich, dass der Updater differenziell ist, was bedeutet, nur die Bereiche auf dem Bildschirm, die sich ändern - vielleicht verursacht durch Bewegung der Sprites - dazu beitragen. Statische Sprites werden nicht neu gezeichnet). Sprites können auf einer der 64 Ebenen vorhanden sein, mit der kleineren Ebene, die näher beim Betrachter ist. Ein Sprite auf Ebene 63 würde knapp über dem Hintergrund liegen, ein Sprite auf Ebene 0 würde über allen anderen Sprites und dem Hintergrund liegen. Die Zeichnenreihenfolge (welches Sprite über welchem gezeichnet wird) ist

unbestimmt, wenn zwei sich überlappende Sprites in der gleichen Ebene liegen.

SP unterstützt vier Arten von Sprites: Maske, OR, XOR und Load. Die Art des Sprites bestimmt, wie das Sprite gezeichnet wird und wie lange es dauert, um es zu zeichnen. Maskierte Sprites sind die langsamsten und verwenden eine Maske, um Teile des Bildschirm-Hintergrunds zu löschen, wo das Sprite erstellt wird (die Maske wirkt wie ein Formstempel, der den Hintergrund herausschneidet, bevor das Sprite hineingezeichnet wird). Mit maskierten Sprites ist es möglich, ein Sprite mit leeren Bereichen zu haben, die durch den Bildschirm überschrieben werden. Or und Xor Sprites sind schneller und verwenden keine Masken. Die Sprite-Grafik eines or/xor Sprites ist logisch OR oder XOR in den Bildschirm verknüpft. Ohne die Verwendung einer Maske, werden leere Bereiche eines Sprites nicht dem zugrunde liegenden Hintergrund löschen. Load Sprites sind mit Abstand die schnellsten, sie werden direkt auf den Bildschirm geschrieben, jeden Hintergrund oder andere Sprites, die unter ihnen liegen können, vollständig auslöschend. .

Die Art der Sprites die Sie wählen ist abhängig von der Leistung und der Wirkung, die Sie benötigen. Fortgeschrittene Programmierer können ein Sprite „modifizieren“, so dass es Teile davon auf verschiedenen Arten gibt. Es kann von Vorteil sein, wegen der Leistung zum Beispiel, die Innenräume der großen Sprites Load-Typ und die Umrisse Masken-Typ zu machen.

Ein Sprite ist eine rechteckige Anordnung von grafischen Zeichen mit einer Zeilenhöhe und Spaltenbreite in Zeicheneinheiten (d.h. 8x8 Pixel-Quadraten) gemessen. Sie können sie sich als große UDGs vorstellen. Sprite Grafiken sind in Spalten definiert: Die grafische Definition für alle Zeichen in einer Spalte muß fortlaufend sein.
Verschiedene Spalten können beliebig im Speicher getrennt werden. Am Anfang lassen Sie uns einen Blick auf eine Beispiels-Definition werfen:
Zunächst einmal, das ist nicht Standard C. Ich bevorzuge es die Sprites im

```
extern uchar col1[];
#asm

._col1
defb @00111100, @00000000
defb @01000010, @00000000
defb @01001010, @00000000
defb @01000010, @00000000
defb @00111100, @00000000
defb @00010000, @10000001
defb @00010000, @11000111
defb @00011000, @11000001

defb @00110100, @00000001
defb @01010000, @00000001
defb @00010000, @00000011
defb @00101000, @00000000
defb @01000101, @00010000
defb @10000010, @00011000
defb @01000000, @00011111
defb @00000000, @11111111

#endasm
```

eingebetteten Assembler zu definieren, so dass ich Binärdaten verwenden kann, um die Grafik besser zu visualisieren. Sie können eine beliebige Standard-C-Variante (wie z.B. ein Array) verwenden, um diese Grafik als Bytes in Ihrem eigenen Programm zu definieren. Was ich hier getan habe ist, ein externes

Array "col1" zu definieren, das in eine Ein-Spalten-Grafik verweist. Da diese Variable extern ist, wird z88dk erwarten, dass es an anderer Stelle als "_col1" definiert ist (der Name mit einem vorangehenden Unterstrich). Mit ein wenig Trickserei arrangierte ich diesen Namen in dem eingebetteten Assembler -Fragment weiter oben, so dass die C-Variable "col1" an die Stelle der eingebetteten Assembler Bezeichnung "_col1" zeigte.

Wie Sie sehen können, ist "_col1" die Position einer wandernden Strichmännchen Grafik. "_col1", wie bei allen Spalten, ist ein Zeichen breit und in diesem Fall zwei Zeichen hoch. Im Gegensatz zu UDGs, die ein einzelnes Byte verwenden, um 8 horizontale Pixel zu definieren, verwenden Sprites zwei Bytes. Das erste Byte definiert die 8 Pixel wie bei UDGs und das zweite Byte definiert die Maske. Die Maske hat 1er, an denen der im Hintergrund liegende Bildschirm durch das Sprite durchscheint, und 0er, wo der im Hintergrund liegende Bildschirm abgedeckt wird. Die 0er in den Masken Bytes im Kopf des Strichmännchens sorgen dafür, dass sein Kopf als leer oval gezeichnet wird, egal welcher Hintergrund hinter dem Sprite liegt. Nur Maskentyp-Sprites beachten diese Maske Bytes. OR, XOR und Load-arten ignorieren die Masken-Bytes. In hi-Color-modus, muss ein drittes Attribut Byte für jedes vertikale Pixel definiert werden.

Um ein Sprite zu erstellen, machen wir einen Anruf zu "sp_CreateSpr":

Ein erfolgreicher Aufruf zu

```
struct sp_SS *man;
man = sp_CreateSpr(sp_MASK_SPRITE, 2,
      col1, 1, TRANSPARENT);
```

"sp_CreateSpr" legt das Sprite aus dem Bild und gibt einen Zeiger auf "struct sp_SS" zurück, welcher die Informationen wie die Größe und die Position des Sprites hält. Sie können einen Blick auf ihre Definition in der "spritepack.h" Header-Datei werfen, für weitere Informationen. Der erste Parameter gibt den Sprite-Typ an, der erstellt werden soll, hier ist es ein maskiertes Sprite, das erstellt wurde. Weitere Optionen sind sp_OR_SPRITE, sp_XOR_SPRITE und sp_LOAD_SPRITE. Der nächste Parameter gibt an, wie groß das Sprite in Zeichen ist. Unser Strichmännchen ist zwei Zeichen hoch. Der dritte Parameter zeigt auf die grafische Definition der ersten Spalte in dem Sprite. Sprites werden durch Hinzufügen einer Spalte nach der anderen gebaut, unser Strichmännchen wird zur Zeit nur eine Spalte haben. Der vierte Parameter ist die Ebene, die das Sprite besetzt. Dies kann eine beliebige Zahl von 0 bis 63 sein, mit 0 als nächste Position zum Betrachter und somit über allen anderen Sprites, und 63 bedeutet, gerade über dem Hintergrund und unter allen anderen Sprites. Der letzte Parameter ist davon abhängig, in welchem Video-Modus wir sind. Im Spectrum-Modus ist dieser letzte Parameter die Farbe der gesamten Sprite-Spalte. Die TRANSPRENT Farbe, wenn Sie sich erinnern, bedeutet, dass die Sprite-Spalte nicht die Farbe auf dem Bildschirm ändert. Es ist nur möglich, eine einzelne Farbe für die gesamte Spalte in diesem Funktionsaufruf anzugeben, aber wir werden später

untersuchen, wie man einzelne Charakter-Quadrate eines Sprites mit anderen Mitteln färben kann. In dem Hi-Res-Modus wird dieser letzte Parameter ignoriert, und in dem Hi-Color-Modus ist dieser letzte Parameter die Farbschwelle.

Sehr gut, wir haben ein Sprite. Wir können dieses Sprite irgendwohin auf dem Bildschirm (oder haraus!) bewegen, mittels zwei Hauptfunktionen: "sp_MoveSprAbs" - bewege Sprite absolut - und "sp_MoveSprRel" - bewege Sprite relativ. Sie können sich wahrscheinlich vorstellen, von den Namen her, die erste wird ein Sprite auf eine bestimmte Pixelposition bewegen, während die letztere sie von ihrer aktuellen Position um einen relativen Pixelabstand vom jetzigen bewegen wird. Wenn Sie sich diese Funktionen in "spritepack.h" ansehen, sehen Sie auch ein paar Variationen. Diese Variationen werden in einem der nächsten Abschnitte über Leistungsverbesserung und in diesem Übersichtstutorial diskutiert werden.

Wie immer, ein Sprite zu bewegen, bewirkt nicht sofort, dass sich die Anzeige ändert. Die Anzeige wechselt nur, wenn "sp_UpdateNow" aufgerufen wird. Sehen Sie ein Beispiel (Vorkompilierte Datei unter: http://www.timexsinclair.org/alvin/spritepack/stickman1.zip)

Sie sollten in der Lage sein, das meiste von diesem Programm zu verstehen,

```
#include <spritepack.h>
#pragma output STACKPTR=61440

extern struct sp_Rect *sp_ClipStruct;
#asm
LIB SPCClipStruct
._sp_ClipStruct         defw SPCClipStruct
#endasm

extern uchar col1[];
uchar hash[] = {0x55,0xaa,0x55,0xaa,0x55,0xaa,0x55,0xaa};
struct sp_UDK keys;

/* Create memory allocator for sprite routines */
void *my_malloc(uint bytes)
{
   return sp_BlockAlloc(0);
}
void *u_malloc = my_malloc;
void *u_free = sp_FreeBlock;

main()
{
   uint reset;
   char dx, dy, i;
   struct sp_SS *man;
   #asm
   di
   #endasm
   sp_TileArray(' ', hash);
```

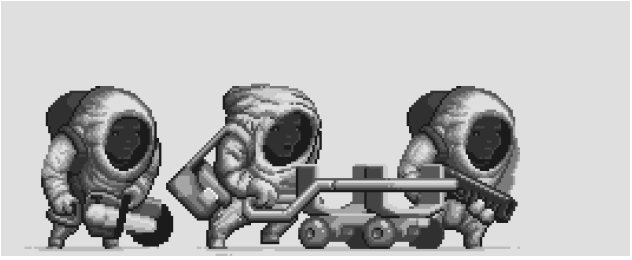
```
   sp_Initialize(INK_WHITE | PAPER_BLACK, ' ');
   sp_Border(MAGENTA);
   sp_AddMemory(0, 255, 14, 0xb000);

   keys.up    = sp_LookupKey('q');
   keys.down  = sp_LookupKey('a');
   keys.left  = sp_LookupKey('o');
   keys.right = sp_LookupKey('p');
   keys.fire  = sp_LookupKey(' ');
   reset      = sp_LookupKey('r');

   man = sp_CreateSpr(sp_MASK_SPRITE, 2, col1, 1, TRANSPARENT);
   sp_MoveSprAbs(man, sp_ClipStruct, 0, 10, 15, 0, 0);
   while(1) {

      sp_UpdateNow();

      i = sp_JoyKeyboard(&keys);
      if ((i & sp_FIRE) == 0) {
         dx = dy = 1;
      } else {
         dx = dy = 8;
      }
      if ((i & sp_LEFT) == 0)
         dx = -dx;
      else if ((i & sp_RIGHT) != 0)
         dx = 0;
      if ((i & sp_UP) == 0)
         dy = -dy;
      else if ((i & sp_DOWN) != 0)
         dy = 0;

      if (sp_KeyPressed(reset))
         sp_MoveSprAbs(man, sp_ClipStruct, 0, 10, 15, 0, 0);
      else
         sp_MoveSprRel(man, sp_ClipStruct, 0, 0, 0, dx, dy);
   }
}
/* Sprite Graphics defined in some inline assembler */
#asm
defb @00111100, @00000000
defb @01000010, @00000000
defb @01001010, @00000000
defb @01000010, @00000000
defb @00111100, @00000000
defb @00010000, @10000001
defb @00010000, @11000011
defb @00011000, @10000001
defb @00110100, @00000001
defb @01010000, @00000001
defb @00010000, @00000011
defb @00101000, @00000000
defb @01000101, @00000000
defb @10000010, @00010000
defb @01000000, @00011000
defb @00000000, @00011111

#endasm
```

aber es gibt ein paar neue Sachen darin. Das am einfachsten zu erklärende ist der benutzerdefinierte Tastatur-Joystick. Die Struktur "sp_UDK keys;" Deklaration schafft eine Struktur, die Scan-Codes für einen Tastatur-Joystick speichert, für links, rechts, oben, unten und einen einzigen Feuerknopf. Die Scancodes sind gespeichert in der Struktur mit Aufrufen zu "sp_LookupKey", die zuvor durchgeführt wurde, als wir daran interessiert waren, zu prüfen, ob einzelne Tasten gedrückt wurden. Ein späterer Aufruf von "sp_JoyKeyboard" liest diesen Tastatur-Joystick und gibt ein einzelnes Byte zurück, welches die aktiven Richtungen auf der Tastatur identifiziert. Der Code zeigt, wie das Byte interpretiert wird.

Speicherzuweisung ist komplizierter zu erklären. Wie Sie sich erinnern können, ist der Updater Zeichenzellen-orientiert. Wenn Sprites erstellt werden, werden sie in zeichen-große Stücke zerteilt und jedes Stück ist wie durch ein "struct sp_CS" (siehe "spritepack.h"-Header -Datei) beschrieben, die wiederum Informationen enthalten, wie das Sprite in jeder Zeichen-Zelle gezeichnet wird. Also ein Aufruf an "sp_CreateSpr" gibt nicht nur eine 14-Byte "struct sp_SS" zurück, das das Sprite als Ganzes beschreibt, sondern hinter den Kulissen erschafft SP ein 14-Byte "struct sp_CS", um jede Zeichen-Zelle zu beschreiben, welche ein Sprite ausmachen. Die Frage ist, wo bekommt die SP-Bibliothek den zusätzlichen Speicher für diese Strukturen her?

Die Antwort ist, dass Sie ein Unterprogramm zur Verfügung stellen müssen, den verfügbaren Speicher auf Anfrage zu reservieren. Dieses Unterprogramm wird durch "u_malloc" (User malloc) angezeigt, dessen voller ANSI-Prototyp "void *(*u_malloc)(uint Bytes);" ist. Auf Deutsch, nimmt die Funktion "u_malloc" einen einzigen unsigned int Parameter wie viele Bytes angefordert werden, und muss die Adresse dieses freien Speichers zurückgeben oder NULL, wenn kein Speicher verfügbar ist. Wenn Sie mit C vertraut sind, werden Sie sehen, dass dies genau das, was der Standard-C "malloc" Anruf macht. Sie können in der Tat die C-Funktion malloc verwenden, durch die Deklaration "void u_malloc = *malloc;" (Hinweis: Die z88dk malloc Umsetzung erfordert, dass Sie ein Array aus dem Speicher als malloced deklarieren - wir erinnern uns es mit einer 64K-Maschine zu tun haben, nicht mit einer Multi-Megabyte-Maschine mit einem virtuellen Adressraum!). Hier ist ein Beispiel: http://www.timexsinclair.org/alvin/spritepack/programmer-intro.htm#exmalloc). Diese zusätzliche Dereferenzierungs= ebene erweitert Optionen, die dem Programmierer zur Verfügung stehen.

Alvin



Wird fortgesetzt...

Anm.: Bitte beachten Sie dass gewisse Sachen bei z88dk in der nächster geplanter Version 1.11 geändert werden, teilweise gilt das bereits für die nightly builds.

sintech

REPARATUR, ZUBEHÖR & ERSATZTEILE

SINTECH ist ein weltweiter Vertrieb — von Hard- und Software für fast alle Systeme. Sie finden uns in Filderstadt, südlich von Stuttgart.
Desweiteren betreiben wir Niederlassungen in Tschechien und in Großbritannien.

Unser Online-Shop ist mit all unseren Produkten versehen. Immer wieder finden Sie bei uns Neuheiten oder Klassiker in der Rubrik Spectrum Hard- und Software.

Wir schwimmen mal gegen den Strom – mal mit. aber stehen immer für Spectrumfreude pur.

Wann schauen Sie vorbei?

sintech
DEUTSCHLAND

SINTECH.DE LTD
Gastäckerstr. 23
70794 Filderstadt
www.sintech-shop.de

sintech
CZECH REPUBLIC

SINTECH.CZ LTD
Masarykova 767
69801 Veseli nad Moravou
www.sintech-shop.cz

sintech
UNITED KINGDOM

SINTECH.UK LTD
1 Moorthen Court, Quedgeley
Gloucester, GL2 4LE
www.sintech-shop.co.uk

SEIT 1994

www.sintech-shop.com